APPENDIX E

11 SHEETS

Merchant Accounts and Key Management

Merchant Accounts

Each merchant has an entry in the principal table in the payment database. This entry is used to contain the merchant name, which is used in the Smart Statement, and the merchant contact information, which is used in the customer service or transaction detail area.

Each merchant will also have an entry in the account table, which account is credited when a payment transaction is processed.

Finally, each merchant will have several keys recorded in the secretkey table. These are the keys which are acceptable for signing payment URLs. One key, stored in the merchant's entry in the principal table, is distinguished, and is used to sign access URLs.

The principal table

The principal table stores the name, address, telephone and fax numbers, and email address of the merchant. This information is revealed to customers in the appropriate setting.

The principal table also contains an access name and password, which are used for enabling changes to the contact information. (And which could be used for access control to the store itself.

The principal table has a secret key stored, which is a copy of one stored in the secretkey table. This distinguished key is used to sign access urls.

The account table

The account table, in the case of a buyer, contains payment credentials: a credit card number, expiration date, and billing address. In the case of a merchant, the content is undefined, but the account number is used to track credits as transactions are made.

The secretkey table

Each merchant has several keys (nominally four) in circulation. The idea is that a key is kept in circulation as long as a payment URL signed with that key is valid. Nominally, payment URLs are re-signed every month or so, on a rotating schedule. Each week, the oldest key is retired and a new one generated. Payment URLs signed with the oldest key are resigned with the new one. The newest key is used to sign access URLs.

Three parties need to know a merchant's keys. The merchant must know, in order to sign payment URLs. The payment system must know, in order to validate payment URLs and to sign access URLs, and the merchant server must know, in order to validate access URLs. (n.b. There is no particular reason to use the same keys for payment and for access.)

If GET style access URLs are used, the merchant server does not need to know the key.

The secret key table contains the key itself, an expiration date, after which the key will not be honored, the principal_id of the owner, and a key_id, which is never reused.

The merchantserver table

In an environment where there are multiple merchant servers, the merchant server table tells the payment system which servers are hosting a particular merchant. The payment system needs to know so that secret keys are distributed to the correct servers. (Not Implemented yet).

Key management

Keys are created by a special utility, ideally one kept in a secure environment. New keys are entered in the secretkey table, and also transported out of band to the merchant server and merchant.

The newest key is entered in the principal table as well as the secretkey table. This is simply a fast path for access to a key to be used to generate the access URL. (Note: The key can only be entered into the principal table record after it has been installed at the merchant server, so the second newest key will typically be there.

In order for the payment system to work, the following steps must be accomplished:

- Create principal account for merchant
- Create keys for merchant
- Load merchant server with merchant keys
- Use merchant's keys to generate payment URLs

Key pre–loading

In order to allow for real–time creation of new merchant accounts, a number of keys are preallocated, along with empty structures in the principal database. The associated keys are sent out of band to the merchant server. These partially completed principal table entries can then be used by Store Builder with the assurance that their access URLs will be immediately honored.

Lawrence C. Stewart <stewart@openmarket.com>
Last modified: Mon Oct 3 13:39:25 1994

Encoding payment orders in URLs

A payment URL looks like this:

```
http://payment.openmarket.com/pay.cgi?hash:field1=value1& field2=value2
```

In any of the field values, the following characters are escaped: '+', '&', '=', ' ', and anything less than 0x20.

The URL fields encode the details of the payment order:

Key id (`kid`)
: This string, in the format . uniquely identifies the merchant and the key used to sign the payment URL.

Target URL (`url`)
: This is the real URL (usually on the merchant's machine) where the client will be redirected after the payment order is processed.

Amount (`amt`)
: This is the amount to charge for access (presumably in US dollars).

Currency (`cc`)
: This is the currency code of the amount. It is a two character code which for national currencies uses the ISO country code. The field is optional, and US will be used if it is missing.

Domain (`domain`)
: This is a string that identifies the domain being granted access to on the merchant's machine. For example, a domain might be dilbert'', which when paid for, would grant access to any document tagged with that domain.

Expiration (`expire`)
: Seconds of access to be granted on payment. This field is optional.

Description (`desc`)
: This is a text description of the item to be put into the buyer's SmartStatement

Signature (`hash`)
: The signature (hash) is computed as the MD5 hash of {key, fieldstring}, where `fieldstring` is everything after the colon. The `key` is a secret key shared between Open Market and the merchant. The signature prevents user tampering of the

payment URL.

Valid until (`valid`)
This is the time, in seconds since January 1, 1970 at which the URL should no longer be accepted for payment. The field is optional, but is honored if present. This field is useful for making limited time offers, or to limit the ability of buyers to save payment URLs and use them later. (Not Implemented yet)

Bill–to (`billto`)
This permits the merchant to specify an account to which the transaction should be billed. The format of the value is .. The principal_id must match the merchant, and the account must be a valid account for that merchant. (Not implemented yet).

Access URL Format (`fmt`)
This permits the merchant to specify the format of access URL to be generated by the payment system. The field is optional. If not present, a

```
http://merchant.foo.com/@hash:field1=value1& field2=value2/real-u
```

format will be used. If the valus is "get", a

```
http://merchant.foo.com/real-url?@hash:field1=value1& field2
```

format will be used

User ID (`id`)
This permits the merchant to request information about the buyer be returned in the access URL. If the field is "id=?" then the access URL will contain "id=" and "name=" where userID is the buyer's principal ID and use name is the buyer's principal_name.

Some *private fields* are generated during the process of taking the user through the confirmation process. These fields should never occur in a merchant's payment URL (we might want to explicitly prevent this from happening, by signing all intermediate URLs with OMI's server secret key):

Pre–confirmation (`preconf`)
The existence of this field specifies that the user has seen the "pre-confirmation" page, which is the page that gives the user an opportunity to open an Open Market account if he doesn't already have one. The `preconf` value is a timeout— the preconfirmation page is displayed if the current time is past the preconf field. (This feature helps prevent people from synthesizing links to the middle of the confirmation sequence).

Post–confirmation (postconf)

The existence of this field specifies that the user has seen the "post–confirmation" page, which is the page shown after the user's identity is known that may include additional confirmation for the payment transaction.

response (resonse)

If this field exists, then this URL is the target of response for a challenge/response confirmation step. The resonse value is the MD5 hash of the expected response value, which should have been entered as the response field for the form POST for the challenge page (whew!). The server computes the MD5 of what the user entered, compares it to what was expected, and completes the transaction if (and only if) they match.

mkid (mkid)

This field can be used during intermediate steps in the payment process, as a place to remember the merchant keyid, while the URL as a whole is signed with the payment server's key.

Creating the URLs

Routines in /omi/payment/payment–httpd/root/lib contain necessary tcl constructing payment URLs.

Two components are necessary for the payment URL. The URL itself, which refers to the nph–payment script on the payment server, and the ticket, which is the signed name–value list.

The URL is usually a constant, it is stored in .../lib/payment.tcl as

```
set paylinkbase "$payment_server_root/bin/nph-payment.cgi?"
```

where $payment_server_root is inherited from .../lib/mall.conf

The ticket is created using .../lib/ticket.tcl by code like the following:

```
  set nv(name1) value1
  set nv(name2) value2
  set ticket [create-ticket $secretkey nv]
```

where name1–value1 and name2–value2 are the name value pairs to be encoded. The procedure create–ticket properly formats and quotes the name value pairs, and signs the result using an MD5 hash of the string with the secretkey. The secretkey is an arbitrary string, but will usually be a 16–hex–digit DES key.

Once the URL and the ticket are available, the payment URL is constructed by concatenation:

```
set paymentURL $paylinkbase$ticket
```

High level creation of payment URLs

/omi/payment/payment-httpd/root/lib/payment.tcl contains a procedure paylink, which uses a command line format to generate payment URLs with all the right fields. The merchant ID, Key ID and secret key are passed in to payment.tcl as environment variables as follows:

```
if [info exists env(SECRET_KEY)] {set secret_key $env(SECRET_KEY)}
if [info exists env(SECRET_KEY_ID)] {set secret_key_id  $env(SECRET_KE
```

The idea is that paylink will be embedded in html documents which are then processed by tcl to produce the final html containing real payment URLs.

```
   [paylink -text "Dilbert (1.00)" -cost 1 -url http://dilbert.com \
        -domain dilbert -desc "Dilbert cartoon" -ttl 20 ]
```

The markup tool takes the details of the payment, and generates a signed URL that points to our payment server.

Open Issues

- See the TODO list

Encoding access in URLs

An access URL looks like this:

```
http://merchant.foo.com/@hash:field1=value1& field2=value2/real-url
```

In any of the field values, the following characters are escaped: '+', '&', '=', ' ', and anything less than 0x20.

The URL fields encode the details of the access:

Expire (`expire`)
: This fields specifies the expiration time for the access, in seconds since January 1, 1970 UTC.

IP Address (`ip`)
: specifies the ip address from which the request must come

Signature (`hash`)
: The signature (hash) is computed as the MD5 hash of {key, fieldstring}, where `fieldstring` is everything after the colon. The `key` is a secret key shared between Open Market and the merchant. The `ipaddr` is the IP address of the client attempting access and `domain` is the access domain. The signature prevents user tampering of the access URL.

The effect is to grant access to a specified domain, from a specified client IP address, for an (optional) period of time.

If desired, any additional information can be encoded in an access URL.

User ID (`id`)
: The unique ID of the buyer, from the field principal_id of the principal table in the payment database. This field will be included in an access URL if the name value pair id=? was in the payment URL.

User Name (`name`)
: The print name of the buyer, from the field principal_name from the principal table in the payment database. This field will be included in an access URL if the name value pair id=? was in the payment URL.

Open Issues

One problem is revocations, where a user pays for access, and then changes his/her mind and we grant a refund. How to we revoke access for the user for the document they already paid for?

One simple solution is log scanning. We do no special work to deny access for refunded access, but if they access the document again, we notice the access in the log and bill them or send them a threating note. (Since we will presumably remove the item from the Smart Statement, it will require special effort on the user's part to save the access URL).

Another problem is "no longer available" pages on the merchant's computer. The user purchases access, and for some reason or another, the accessed pages are not available, then there needs to be some way for the merchant to reverse the payment.

Shopping Cart Mechanism

how to write payment and shopping cart URLs

These two types of URLs are very similar:

```
Payment URL:
        http://payment.openmarket.com/bin/nph-payment.cgi?TICKET

Add to shopping cart URL:
        http://payment.openmarket.com/bin/nph-cl.cgi?CARTTICKET

Here's what a TICKET is:

A TICKET is a signed collection of name-value pairs.  The way I build
these is with a routine in /omi/httpd/payment/root/lib/ticket.tcl
called "sec_create_ticket" which takes two tcl arrays as arguments.

        sec_create_ticket key_array value_array

The key_array contains the following:
        set key_array(principal_id) xxx
        set key_array(secretkey_id) yyy
        set key_array(secret_key) zzz
where
        xxx is the "principal_id" of a merchant.  This is a record ide
from the database.
        yyy is the identification number of a key in the key database
        zzz is the 16-hex-digit DES key itself.

sec_create_ticket uses the principal_id and the secretkey_id to
add a field like "kid=xxx.yyy" to the ticket.  The secret_key is used
build the digital signature itself.

The value_array contains the necessary name-value pairs to handle paym

        set value_array(domain)  ...domain...  (a domain is a security
                of the thing purchased, it is also used to detect dupl
                purchases, so it should be unique to separately buyabl
        set value_array(desc)    ...item description...
        set value_array(amt)     ...monetary amount...
        set value_array(cc)      ...currency code... (OPTIONAL, but US
        set value_array(expire)  ...seconds of access... (OPTIONAL, de
                                      is 30 days at the moment)
        set value_array(url)     ...URL of the thing bought...
                (In the shopping cart case, this should be the URL of
                the relevant catalog page)
```

Here's what a CARTTICKET is:

A CARTTICKET is the same as a TICKET except that it may also contain a "details=DETAILS" field.

A DETAILS is an escaped name/value string constructed by calling

url_unparse ARRAY

where url_unparse is defined in /omi/httpd/payment/root/lib/ticket.tcl and ARRAY is a tcl array with items in it:

```
        set a(size) XL
        set a(color) blue
        url_unparse a
```

This returns things like: color=blue&size=XL
The details item is just carried along through the shopping cart and ev
entered into the order that goes to the merchant's order entry system.

A CARTTICKET may also contain an aurl field, which is the access URL to use after an item has been purchased (This is really the same as the url field of a payment link.)

Lawrence C. Stewart <stewart@openmarket.com>
Last modified: Mon Oct 3 13:31:23 1994